Orientation in Objects

WS-SecureConversation

) Schulung)

) Beratung)

) Entwicklung)

) Artikel)

AUTOR

Fabian Fassott
Orientation in Objects GmbH

Veröffentlicht am: 24.1.2009

WS-SECURECONVERSATION

Das Tutorial beschreibt die Erstellung eines Web Services, der Benutzer per UsernameToken authentifiziert und dann per WS-SecureConversation Session-Keys zur Gewährleistung von Vertraulichkeit und Integrität erzeugt, um den nachfolgenden Nachrichtenfluss zu sichern.

Orientation in Objects GmbH

Weinheimer Str. 68
D-68309 Mannheim

Tel. +49 (0) 6 21 - 7 18 39 - 0
Fax +49 (0) 6 21 - 7 18 39 - 50

www.oio.de info@oio.de

WS-SECURECONVERSATION

Häufig müssen Nachrichten zwischen Client und Web Service verschlüsselt und signiert werden. Hierfür existiert WS-Security [1]. Für eine einzeln zu sichernde Nachricht kann ein zufälliger Schlüssel generiert werden und beispielsweise mit dem Public Key-Zertifikat des Web Services verschlüsselt übertragen werden. Dies geschieht für jede einzelne Nachricht erneut. Public Key-Kryptographie ist jedoch langsam. Symmetrische Verfahren sind um ein Vielfaches schneller. Hier setzt WS-SecureConversation (WS-SC) [2] an. Mit WS-SC können mehrere Nachrichten durch die gleichen symmetrischen Schlüssel geschützt werden. Der Aufwand durch die Verwendung von Public Key-Kryptographie wird minimiert.

Die Aushandlung der Session Keys (bspw. einer zum Verschlüsseln und einer zum Signieren, einer für beides geht jedoch auch) kann auf unterschiedlichen Wegen erfolgen, allerdings wird als Grundlage immer WS-Trust [3] verwendet. Der Client kann einen Vorschlag machen, den der Service akzeptiert, beide können Schlüsselmaterial liefern oder der Service alleine bestimmt einen Key. Dieser ausgehandelte Key wird mit einem SecurityContextToken assoziiert. Dieses wird in jeder Anwendungsnachricht im SOAP-Header mitgeschickt, um dem Server den zu verwendenden Schlüssel mitzuteilen. Typischerweise hat es den folgenden Aufbau:

```
<SecurityContextToken wsu:Id="SomeId">
  <Identifier>uuid:...</Identifier>
</SecurityContextToken>
```

Beispiel 1: Typisches SecurityContextToken

Mit dem in WS-Security definierten wsu:Id-Attribut kann dem SecurityContextToken zwecks späterer Verweise auf dieses Token eine Id gegeben werden. Das Identifier-Element ist hingegen wie oben erläutert dazu gedacht, die Verbindung zum Schlüssel herzustellen.

Im Folgenden wird erläutert, wie mittels Metro [4], GlassFish [5] und NetBeans [6] zwischen Client und Web Service ein Session Key gemäß WS-SC mit vorheriger UsernameToken-Authentifizierung ausgehandelt werden kann. Dabei werden Grundlagen im Umgang mit den angesprochenen Produkten vorausgesetzt, weswegen nicht jeder einzelne Schritt erklärt wird. Zur Einführung in die Entwicklung mit Metro im GlassFish sind die detaillierteren Beschreibungen des folgenden Artikels über die UsernameToken-Authentifizierung [7] zu empfehlen.

SERVICE-IMPLEMENTIERUNG

Es ist ein Web Service zu erstellen, der wie unter [7] beschrieben UsernameToken-Authentifizierung ausführt. Anschließend muss WS-SC aktiviert werden: Nachdem Username Authentication with Symmetric Key gewählt wurde, sind unter Configure... die auf dem folgenden Screenshot gezeigten Einstellungen vorzunehmen.

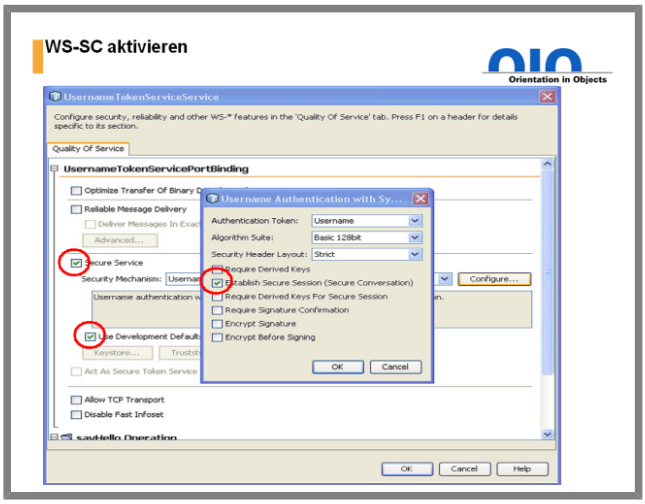

Abbildung 1: WS-SC aktivieren

Der Web Service erwartet jetzt das in Abbildung 2 gezeigte Kommunikationsmuster.

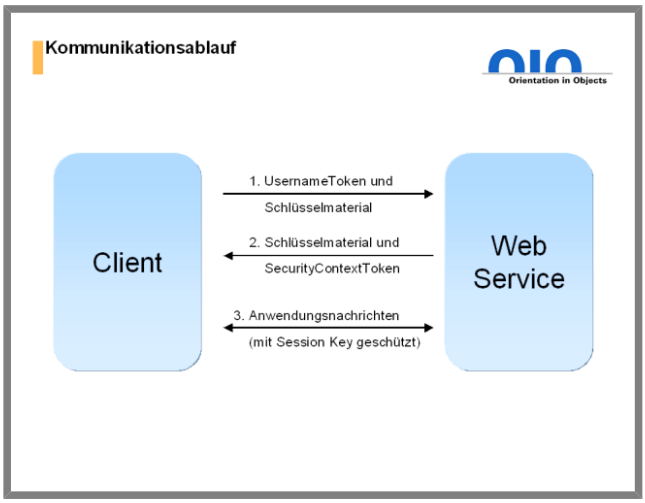

Abbildung 2: Kommunikationsablauf

In SOAP stellt sich der Schlüsselaustausch dann wie in den Beispielen 1 und 2 dar (das UsernameToken wird verschlüsselt im Header übertragen und aus Platzgründen nicht gezeigt):

```
<S:Body wsu:Id="_5007">
  <RequestSecurityToken ...>
    <TokenType>
      http://schemas.xmlsoap.org/ws/2005/02/sc/sct
    </TokenType>
    <RequestType>
      http://schemas.xmlsoap.org/ws/2005/02/trust/Issue
    </RequestType>
    <Entropy xmlns:ns7="..." ns7:Type="BinarySecret">
      <BinarySecret Type".../Nonce">
        OFwU6/UWnGQrZ+PU6MEnQg==
      </BinarySecret>
    </Entropy>
    <KeySize>
      128
    </KeySize>
    <KeyType>
      http://schemas.xmlsoap.org/ws/2005/02/trust/SymmetricKey
    </KeyType>
    <ComputedKeyAlgorithm>
      http://schemas.xmlsoap.org/ws/2005/02/trust/CK/PSHA1
    </ComputedKeyAlgorithm>
  </RequestSecurityToken>
</S:Body>
```

Beispiel 2: Schlüsselmaterial vom Client

Über TokenType und RequestType wird angezeigt, dass ein SecurityContextToken neu zu erstellen ist. Im Entropy-Element sendet der Client sein Schlüsselmaterial. KeySize bestimmt die Schlüssellänge und der erzeugte Key soll für symmetrische Kryptographie verwendet werden. Im ComputedKey-Element schlägt der Client einen Algorithmus vor, mit dem aus dem beidseitigen Schlüsselmaterial der Session Key zu erzeugen ist.

```
<S:Body wsu:Id="_5006">
  <RequestSecurityTokenResponse ...>
    <TokenType>
      http://schemas.xmlsoap.org/ws/2005/02/sc/sct
    </TokenType>
    <RequestedSecurityToken>
      <ns5:SecurityContextToken
      ns2:Id="uuid-ef037748-908c-4630...">
        <ns5:Identifier>
          urn:uuid:fc5178c6-361b-473b-bc32-0af88c82b687
        </ns5:Identifier>
      </ns5:SecurityContextToken>
    </RequestedSecurityToken>
    ...
    <RequestedProofToken>
      <ComputedKey>
        http://schemas.xmlsoap.org/ws/2005/02/trust/CK/PSHA1
      </ComputedKey>
    </RequestedProofToken>
    <Entropy xmlns:ns7="..." ns7:Type="BinarySecret">
      <BinarySecret Type=".../Nonce">
        M9G1A1Mboj17EZXCqozBbw==
      </BinarySecret>
    </Entropy>
    <Lifetime>
      <ns2:Created>
        2008-09-25T08:37:48.015Z
      </ns2:Created>
      <ns2:Expires>
        2008-09-25T18:37:48.015Z
      </ns2:Expires>
    </Lifetime>
  </RequestSecurityTokenResponse>
</S:Body>
```

Beispiel 3: Schlüsselmaterial Server und SecurityContextToken

Der Server gibt das angeforderte SecurityContextToken zurück und schränkt dessen Gültigkeit ein. Das serverseitige Schlüsselmaterial ist ebenfalls in der Antwort enthalten und der angeforderte Session Key-Generationsalgorithmus wird bestätigt.

Unverschlüsselte Übertragung

Werden serverseitig die gleichen Einstellungen wie unter [7] beschrieben vorgenommen, erfolgt auch hier die Kommunikation zwischen Client und Web Service verschlüsselt, wodurch die in den Beispielen 1 und 2 gezeigten Elemente nicht sichtbar sind. Dies kann durch manuelle Anpassungen in der von Metro generierten Service-Konfigurationsdatei geändert werden: Rechtsklick auf das Service-Projekt -> Build. Dann in das Files-Tab wechseln und den Projekt-Ordner expandieren. Unter build\web\WEB-INF\wsit-NAME_DER_IMPL.xml sind dann alle EncryptedParts-Elemente unter dem BootstrapPolicy-Element zu entfernen und der Service zu deployen.

CLIENT-IMPLEMENTIERUNG

Der benötigte Client wird analog zum Client in [7] erstellt. Es kommt der gleiche JAX-WS-Handler zur Übertragung des UsernameTokens zum Einsatz. Die Verwendung von WS-SC wirkt sich nicht auf die Anwendungslogik aus. Mit der Hilfe eines einfachen http-Monitors kann auch nachvollzogen werden, dass für die unterschiedlichen Nachrichten der gleiche Session Key verwendet wird.(Hier sei wieder auf das Tool TCPMon [8] verwiesen.) Um dies zu testen, ist einfach der Web Service mehrmals hintereinander, anstatt nur einmal, aufzurufen.

FAZIT

Mit NetBeans, GlassFish und Metro ist es sehr einfach möglich, einen Web Service mittels UsernameToken zu sichern, einen Session Key gemäß WS-SC auszuhandeln und einen passenden Client zu erstellen. Die Aktivierung von WS-SC erfolgt dabei durch einfache Konfiguration innerhalb der NetBeans IDE. Client-seitig ist für die Übertragung des UsernameTokens lediglich ein einfacher JAX-WS-Handler wie unter [7] beschrieben notwendig. Mehrere Anwendungsnachrichten können dann unter Vermeidung von Public Key-Kryptographie effizient mit symmetrischen Verfahren geschützt werden.

REFERENZEN

- [1] Web Service Security (WSS)
  (http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf)
- [2] WS-SecureConversation
  (http://docs.oasis-open.org/ws-sx/ws-secureconversation/v1.3/ws-secureconversation.html)
- [3] WS-Trust
  (http://docs.oasis-open.org/ws-sx/ws-trust/v1.3/ws-trust.html)
- [4] Metro
  (https://metro.dev.java.net/)
- [5] GlassFish
  (https://glassfish.dev.java.net/downloads/v2ur2-b04.html)
- [6] NetBeans
  (http://www.netbeans.org/downloads/index.html)
- [7] Web Services mit UsernameToken sichern
  (http://www.oio.de/public/xml/usernametoken-tutorial-webservice-security-artikel.htm)
- [8] TCPMon
  (http://ws.apache.org/commons/tcpmon/)